Homo- and Catamorphisms, Reductions and Maps
An Overview

Maarten M Fokkinga
Centre for Mathematics and Computer Science, Amsterdam
University of Twente, Enschede

Version of August 20, 1990

Abstract

The classical notion of *homomorphism*, the Bird-Meertens notion of homomorphism — here called *catamorphism*— having *reduction* and *map* as a special case, are defined formally and investigated with respect to their calculational properties. All this is done both in terse category theoretic terminology, as well as spelled out in more conventional, less abstract, concepts. This note reflects my current understanding of other people's work, in particular the ideas of Lambert Meertens.

1 Introduction

This note grew out of an attempt to record an exposition by Lambert Meertens on the notion of paramorphism, that he had just invented. In doing so it grew somewhat more than I had expected.

Writing notes like this one is just my own way of studying. No result in this paper is my own invention; almost everything has already appeared elsewhere. In particular, my knowledge on this material comes from discussions, lectures and papers by Roland Backhouse, Johan Jeuring, Grant Malcolm, Lambert Meertens, Nico Verwer, Jaap van der Woude, Hans Zantema, and other members of the STOP Algorithmics Club. All errors are mine (and I welcome any comments the reader may have: it would help me learning more about this stuff). Since I myself am quite pleased with the formulations used in this paper, I hope that other people might profit from this writing; hence my motivation for distributing it.

Two classes of functions play a very important rôle in the Bird-Meertens style of programming: homomorphisms and catamorphisms. Roughly speaking, a function f is a *homomorphism* if it "commutes" with the operations (with respect to which it is a homomorphism):

$$f\,(\psi\,x) \quad = \quad \phi\,(f\,x)\,.$$

This is denoted $f : \phi \leftarrow \psi$, assumed that ϕ and ψ take one argument. If, in addition, we take ψ to be the *constructor(s)*, $\underline{\mathsf{in}}$ say, of a data type, we say that f is a *catamorphism*, and can denote this also by $f = (\!|\phi|\!)$. These functions play an important rôle, due to their nice calculational properties ((conditional) functional identities, as given by for example the Uniqueness Property and Promotion Theorem). Special forms of catamorphisms are the well-known *reductions* and *maps*. The Map Distribution law and join-Reduction Promotion

law are unconditional, hence useful, functional identities. Next to the concepts of homomorphism and catamorphism, Meertens[7] has introduced the concept of *paramorphism*, that is, a function satisfying equations of the form

$$f\,(\underline{\mathsf{in}}\,x) \quad = \quad \phi(f\,x,\,x)\,.$$

The factorial function is an example: $fac\;x \;=\; ...(fac\;x)...x...$. These functions are not necessarily catamorphisms, but occur rather often, as you will agree. We refer the interested reader to Meertens[7].

We formally define the above concepts, as well as the so-called initial data types, in a very general way; e.g., we define maps for any initial data type. To be able to do so, we shall use Category Theory terminology. We leave only one theorem unproven —in this paper— and all the proofs turn out to be very simple short calculations. (This was for me quite unexpected; it has taken me one and a half year to understand what turns out to be so simple!)

As a warming up we begin by recapitulating the classical notions of *algebra* and *homomorphism* in Section 3; a $\dagger$-algebra is a pair (A,ϕ) where $\phi : A \leftarrow A^\dagger$, and a homomorphism to (A,ϕ) from (B,ψ) is a function $f : A \leftarrow B$ with $\phi \cdot f^\dagger = f \cdot \psi$, denoted $f : \phi \xleftarrow{\dagger} \psi$. Here, $\dagger$ is a functor that gives the source type of ϕ , namely $A^\dagger$, and of ψ , namely $B^\dagger$. Using disjoint union and cartesian product and the trivial one point set, we are able to assume that any algebra has precisely one carrier, precisely one operation, which in turn has precisely one parameter. If this is well known to you, you may skip this section without loss of continuity. Then, in Section 4 we define the notion of *initial data type* and *catamorphism*; the former is written $(L,\underline{\mathsf{in}}) = \mu(\dagger)$ and the latter $(\!|\phi|\!)_\dagger : A \leftarrow L$ for given function $\phi : A \leftarrow A^\dagger$. Again, if this sounds familiar to you, you may savely skip most of this section. The notion of reduction and map get their turn in Section 5. We generalise what we know for join-list to the general case; in particular the factorisation of catamorphisms as reductions after a map, as well as the join-reduction promotion law are derived. You do not miss any new law if you skip this section; what remains is to enjoy the notational idiosyncrasies explained in the next section.

2 Notation

On right to left reading

For ease of memorisation *and manipulation* we shall be very systematic in the way we write equations and assertions. We choose to write an application of a function to an argument in the conventional order: the function (and output) at the left and the argument (input) at the right. Therefore we shall *everywhere* write things that have to do with the source at the right, and things that have to do with the target at the left. (Moreover, we choose letters in a formula in alphabetical order of (normal reading) appearance.) Thus we write $f \cdot x$, $f : A \leftarrow B$, $\quad f : (A,\phi) \leftarrow (B,\psi)$, $\quad f : \phi \leftarrow \psi$, $\quad \phi \cdot f^\dagger = f \cdot \psi$, and (taking $\psi := \underline{\mathsf{in}}$) $\phi \cdot f^\dagger = f \cdot \underline{\mathsf{in}}$. It will be explained that, reading *from right to left*, the latter formula can be seen as a recursive definition of f on a data type with $\underline{\mathsf{in}}$ as constructor. So, the consequence of the consistency (taking the conventional order in an application for granted) is that most equations had better been read from right to left. Anyway, equality is a symmetric relation, so nobody can object formally.

Having finished the writing of this paper, I think that it may be more convenient not to follow the conventional order within an application, and to exchange everywhere the position

of source and target. The only drawback is that we have to write $x \cdot f$ instead of the familiar $f \cdot x$. This would restore the conventional order within a definition and of the arrows, and has the additional advantage that the right to left direction of writing coincides with the direction from the source to target.

Combinators

By way of experiment, we use Meertens' latest suggestion for a notation for cartesian product, disjoint union, and related functions. The symbols have been chosen with the following goal in mind:

- they should be placed in infix position;
- they should neither use commas and parentheses, nor the familiar times and plus symbol, because these symbols are too important to waste for this specific use;
- they should reflect the duality between the two concepts;
- they should have a mnemonic value, easy to write and to TEX.

The symbols for selection and injection are mine; they are just the "natural symbolification" of the combinators they denote. (I do not know good short names that fit the symbols.) It remains to be seen whether the suggested symbols will be satisfying in practise.

Here is a list of the new combinator notations, together with the familiar notation and some explanation to the right.

("parallel"?)	$A \parallel B$	$A \times B$	cartesian product
("parallel"?)	$f \parallel g$	$f \times g$	$(\lambda\langle x, y\rangle :: \langle f\, x,\ g\, y\rangle)$
("sharing"?)	$f \Uparrow g$	(f, g)	$(\lambda x :: \langle f\, x, g\, x\rangle)$
"left one"	$\ll$	$\pi_1,\ \ll$	$(\lambda\langle x, y\rangle :: x)$
"right one"	$\gg$	$\pi_2,\ \gg$	$(\lambda\langle x, y\rangle :: y)$
("exclusive parallel"?)	$A \Upsilon B$	$A + B$	disjoint union
("exclusive parallel"?)	$f \Upsilon g$	$f + g$	$(\lambda z ::\ \mathbf{case}\ z\ \mathbf{of}\ \mathbf{inl}\, x : \mathbf{inl}\, f\, x,\ \mathbf{inr}\, y : \mathbf{inr}\, g\, y)$
("??")	$f \Downarrow g$	$[f, g]$	$(\lambda z ::\ \mathbf{case}\ z\ \mathbf{of}\ \mathbf{inl}\, x : f\, x,\ \mathbf{inr}\, y : g\, y)$
"inleft"	$\curlywedge$	$\mathbf{inl},\ \jmath_L$	$(\lambda x :: \mathbf{inl}\, x)$
"inright"	$\curlyvee$	$\mathbf{inr},\ \jmath_R$	$(\lambda y :: \mathbf{inr}\, y)$

For simplicity, we consider the combinators $\parallel$, $\Uparrow$, Υ and $\Downarrow$ to be binary ($\ll$, $\gg$, $\curlywedge$, and $\curlyvee$ are —at this functional level— nullary combinators: they take no functions as argument, but are themselves a complete term). Following Meertens[6], we could interpret them as associative, n-ary combinators, which makes practical use much easier.

Besides the above combinators we use also the symbol $\cdot$ and interpret it as (functional) composition. The symbol $\cdot$ has the weakest binding power of all; then come the above binary combinators. All the other operators have a higher binding power than these combinators.

Meertens[6] gives a combinatorial interpretation of these combining forms: a term denotes a box, with subterms denoting subboxes; elementary boxes are the given functions and the nullary combinators. The boxes are wired together: $\cdot$ connects the output lines of its right argument to the input lines of its left argument. Considering the values to flow top-down, the

symbols suggest exactly the wiring; e.g., $f \parallel g$ consists of subboxes f and g placed besides each other, and their input lines together form those of the whole box, and similarly for the output lines. The combinator $\Uparrow$ splits the input wire into two, duplicating the value on the line and feeding it into both boxes. The symbol $\sim$ placed on two parallel wires means that only one of the lines carries a value at a time: thus the usual "tags" that make the values disjoint, are encoded here by the geometrical position that the value has. The combinator $\Downarrow$ lets the two (exclusive) wires coalesce together.

The input and output ports of a box in a system are labelled by the set from which the values are drawn. For a box f we denote this by $f : A \leftarrow B$, where A labels the output port and B the input port. Both A and B may be composite, e.g., $\mathbb{N} \parallel \mathbb{N}$. Thus there is a well-typing requirement; this is formalised as follows:

$$\Leftarrow \frac{f \parallel g \ : \ A \parallel B \leftarrow C \parallel D}{f : A \leftarrow C \quad g : B \leftarrow D} \qquad \Leftarrow \frac{f \Uparrow g \ : \ A \Uparrow B \leftarrow C \Uparrow D}{f : A \leftarrow C \quad g : B \leftarrow D}$$

$$\Leftarrow \frac{f \Upuparrows g \ : \ A \parallel B \leftarrow C}{f : A \leftarrow C \quad g : B \leftarrow C} \qquad \Leftarrow \frac{f \Downarrow g \ : \ A \leftarrow B \Uparrow C}{f : A \leftarrow B \quad g : A \leftarrow C}$$

$$\Leftarrow \frac{\ll \ : \ A \leftarrow A \parallel B}{} \qquad \Leftarrow \frac{\curlywedge \ : \ A \Uparrow B \leftarrow A}{}$$

$$\Leftarrow \frac{\gg \ : \ A \leftarrow B \parallel A}{} \qquad \Leftarrow \frac{\curlyvee \ : \ A \Uparrow B \leftarrow B}{}$$

Rather than explaining the semantics in the above lambda-formulation, we list here the laws that we postulate to hold for the combining forms and specific functions. Each law is entirely evident from the combinatorial interpretation of the combinators. The laws are just functional identities, and so fit entirely in the categorical treatment that we shall follow in the sequel. [1] We shall refer to these laws collectively as "combinator calculus". Notice the full duality between the notation, syntax and semantics between the left and right column.

$$\begin{array}{rcl}
f \parallel g \cdot h \Upuparrows j & = & (f \cdot h) \Upuparrows (g \cdot j) \\
f \parallel g \cdot h \parallel j & = & (f \cdot h) \parallel (g \cdot j) \\
f \Upuparrows g \cdot h & = & (f \cdot h) \Upuparrows (g \cdot h) \\
\ll \cdot f \parallel g & = & f \cdot \ll \\
\gg \cdot f \parallel g & = & g \cdot \gg \\
\ll \Upuparrows \gg & = & \mathsf{id} : A \parallel B \leftarrow A \parallel B \\
f \parallel g = h \parallel j & \equiv & f = h \wedge g = j \\
f \Upuparrows g = h \Upuparrows j & \equiv & f = h \wedge g = j
\end{array}
\qquad
\begin{array}{rcl}
f \Downarrow g \cdot h \Uparrow j & = & (f \cdot h) \Downarrow (g \cdot j) \\
f \Uparrow g \cdot h \Uparrow j & = & (f \cdot h) \Uparrow (g \cdot j) \\
f \cdot g \Downarrow h & = & (f \cdot g) \Downarrow (f \cdot h) \\
f \Uparrow g \cdot \curlywedge & = & \curlywedge \cdot f \\
f \Uparrow g \cdot \curlyvee & = & \curlyvee \cdot g \\
\curlywedge \Downarrow \curlyvee & = & \mathsf{id} : A \Uparrow B \leftarrow A \Uparrow B \\
f \Uparrow g = h \Uparrow j & \equiv & f = h \wedge g = j \\
f \Downarrow g = h \Downarrow j & \equiv & f = h \wedge g = j
\end{array}$$

The identity laws can also be motivated as follows. In a few steps one can prove from the other laws that $\ll \Upuparrows \gg \cdot \ f \parallel g \ = \ f \parallel g \cdot \ll \Upuparrows \gg$. In view of the well typing requirements this equality has been strengthened by postulating that $\ll \Upuparrows \gg$ is the identity for $\parallel$-types;

[1] Added in proof: if we simply postulate that $(\parallel, \Upuparrows, \ll, \gg)$ form a "categorical product", and define $f \parallel g = (\ll \cdot f) \Upuparrows (\gg \cdot g)$, then the laws are all derivable. Similarly for $(\Uparrow, \Downarrow, \curlywedge, \curlyvee)$ and "categorical co-product". See Malcolms thesis [4].

similarly for $\Updownarrow$. A useful corollary of the identity laws is that for any $f : A \leftarrow B \Updownarrow C$ one has $f = g \Updownarrow h$ where $g = f \cdot \upharpoonleft$ and $h = f \cdot \upharpoonright$; and similarly for $\|$.

The one point set is denoted $\mathbb{1}$. The set $\mathbb{1}$ being one point means that for any A there exist exactly one function into $\mathbb{1}$, denoted ι_A:

$$\Leftarrow \frac{f = \iota_A}{\text{(for } f : \mathbb{1} \leftarrow A \text{)}} \qquad \Leftarrow \frac{\iota_A : \mathbb{1} \leftarrow A}{}$$

3 Algebra and Homomorphism Categorically described

Recall the notion of category: a *category* consists of

- a collection of so-called *objects* (A, B, C, ..., X),
- a collection of so-called *arrows* ($f, g, h, \ldots, \phi, \psi, \ldots$), each arrow going *to* an object (called the *target*) *from* an object (called the *source*) which is denoted $f : A \leftarrow B$,
- and a so-called *composition* (denoted $\cdot$) of arrows (composing an arrow $A \leftarrow B$ with an arrow $B \leftarrow C$ into an arrow $A \leftarrow C$) that is associative and has for each object an identity $\mathsf{id}_A : A \leftarrow A$.

A term $f \cdot g$ in which the source of f differs from the target of g is meaningless; all the identities that are asserted assume that the terms involved are meaningful. Moreover, if no confusion can result, knowing that sources and targets match, we will omit any indication of the sources and targets of the arrows, and write e.g. id instead of id_A. We could now formalise the above definition into a few inference rules (two for the typing, three equations). However, these laws are so basic that we shall never, nowhere, refer to them explicitly. Associativity of $\cdot$ is even built in into the notation by dropping the parentheses.

An object A is *initial* in a category if for any object B in the category there exists precisely one arrow $B \leftarrow A$.

A *(mono-)functor* $\dagger$ is a mapping (written as a superscript) from objects to objects, and from arrows to arrows such that

- $f^\dagger : A^\dagger \leftarrow B^\dagger$ whenever $f : A \leftarrow B$, and
- $\mathsf{id}_A^\dagger = \mathsf{id}_{A^\dagger}$ and $(f \cdot g)^\dagger = f^\dagger \cdot g^\dagger$.

This notion can be generalised to functors of several arguments. In particular, a *bi-functor* $\dagger$ is a mapping (written as a normal infix operation), mapping two objects into an object, and two arrows into an arrow, in such a way that

- $f \dagger g : A \dagger C \leftarrow B \dagger D$ whenever $f : A \leftarrow B$ and $g : C \leftarrow D$, and
- $\mathsf{id}_A \dagger \mathsf{id}_B = \mathsf{id}_{A \dagger B}$ and $(f \cdot g) \dagger (h \cdot j) = f \dagger h \cdot g \dagger j$.

Examples of bi-functors are: $\|$ and $\Updownarrow$. We can consider any object A itself as a mono-functor by defining $X^A = A$ and $f^A = \mathsf{id}_A$. When A^* denotes the set of lists over A and f^* the usual f-map, then, in fact, $*$ is a mono-functor; all requirements of a functor are satisfied by $*$. We return to maps in Section 5.

We intend to interpret objects as just sets of elements (called *types*) and arrows as *typed functions*, mapping elements onto elements, having normal function composition as the arrow

composition; and we shall use this terminology most of the time. The intended interpretation is the so-called category **SET**. When we do not use this knowledge in a proof, we say that the proof is *categorical*. All of the proofs in this paper are categorical. In fact, each equational step is justified by identities from the combinator calculus, the category calculus (implicit), the functor calculus, or μ-type calculus (just a single identity), plus of course derived and definitional identities.

* * *

Let $\dagger$ be a functor. Then a $\dagger$-*algebra* is a pair (A, ϕ) where A is a type and ϕ is an arrow $\phi : A \leftarrow A^\dagger$. Thus an algebra has one carrier set, one operation, and the operation takes only one argument. To explain that this is without loss of generality, we shall show how

$$(\mathbb{N};\ 0 : \mathbb{N},\ 1 : \mathbb{N},\ + : \mathbb{N} \leftarrow \mathbb{N} \parallel \mathbb{N},\ \times : \mathbb{N} \leftarrow \mathbb{N} \parallel \mathbb{N})$$

can be described as a $\dagger$-algebra in the above sense. First replace the 0-ary operations by 1-ary operations, having the one point type $\mathbb{1}$ as their source: $0^\bullet : \mathbb{N} \leftarrow \mathbb{1}$ and $1^\bullet : \mathbb{N} \leftarrow \mathbb{1}$. Then define $\dagger$ such that $\mathbb{N}^\dagger$ is the disjoint sum of the source types of the operations, so for arbitrary X

$$X^\dagger \ = \ \mathbb{1} \Downarrow \mathbb{1} \Downarrow (X \parallel X) \Downarrow (X \parallel X)$$

and take ϕ as the "case-amalgamation" of the operations:

$$\phi \ = \ 0^\bullet \Updownarrow 1^\bullet \Updownarrow + \Updownarrow \times \qquad : \mathbb{N} \leftarrow \mathbb{N}^\dagger .$$

Now $(\mathbb{N};\ 0, 1, +, \times)$ is modeled by $(\mathbb{N}, \phi)$.

Also many-sorted algebras can be modeled by a $\dagger$-algebra (A, ϕ). As an example, consider $(\mathbb{N}, \mathbb{B}; \ldots, \leq, \ldots)$. Now we take $A := \mathbb{N} \Downarrow \mathbb{B}$, thus representing naturals in the left branch of A: $repr_{\mathbb{N}} = \downarrow$ and similarly $repr_{\mathbb{B}} = \uparrow$. Then, like we did above, $\phi := \ldots \Updownarrow leq \Updownarrow \ldots$. Operation leq must have type $A \leftarrow A \parallel A$, and should model operation $\leq$. That is, the equation

$$leq \cdot repr_{\mathbb{N}} \parallel repr_{\mathbb{N}} \quad = \quad repr_{\mathbb{B}} \cdot \leq \quad \textbf{on } \mathrm{Domain}(\leq)$$

should hold true. It is easy to argue that such operations leq exist; hence a $\dagger$-algebra exists that models $(\mathbb{N}, \mathbb{B} : \ldots, \leq, \ldots)$. It is, however, not quite trivial to express leq in terms of $\leq$ and combinators like $\parallel$, $\Uparrow$ and so on. I see no way to do it without curried functions; i.e., I have to suppose that the category is cartesian closed. Here is a possible definition of leq: (false stands for "type error": an element $repr_{\mathbb{B}}(x)$ is recieved whereas some $repr_{\mathbb{N}}$ is expected)

$$leq \ = \ \mathrm{apply} \cdot ((\lambda x :: repr_{\mathbb{B}} \cdot (x \leq) \Updownarrow \mathsf{false}^\bullet) \Updownarrow (repr_{\mathbb{B}} \cdot \mathsf{false}^\bullet)) \parallel \mathsf{id} .$$

So, in a $\dagger$-algebra (A, ϕ), the functor $\dagger$ plays the rôle of the type information of what is usually called the signature, and ϕ is the amalgamation of all the operations of the algebra. Thus any algebra can be considered to have one, unary, operation whose source type structure is given by a single functor. Needless to say that this description is very suited for theoretical discussions and proofs about algebras, but is less suited for actual calculations within the algebra.

Let $\dagger$ be a functor, and (A,ϕ) and (B,ψ) be two $\dagger$-algebras. A function $f : A \leftarrow B$ is called a $\dagger$-*homomorphism* to (A,ϕ) from (B,ψ) if

$$\phi \cdot f^\dagger \;=\; f \cdot \psi$$

which we denote by

$$f : \phi \mathrel{\dot{\dashv}} \psi \,.$$

When read from right to left, the equation expresses that doing f after operation(s) ψ of the source algebra gives the same result as first doing $f^\dagger$ and then the operation(s) ϕ of the target algebra. Notice moreover that we can define $\dagger$ by $X^\dagger = X \parallel X$ and $f^\dagger = f \parallel f$, and similarly for other functors like $\Downarrow$, $*$, and constant functors. If we do so, then doing $f^\dagger$ in $B^\dagger$ actually means doing f on the ingredients separately. Hence this notion of homomorphism agrees with the notion that we know for the familiar, conventional presentation of algebra.

It is easy to show that homomorphisms compose nicely:

$$\Leftarrow \frac{f \cdot g : \phi \mathrel{\dot{\dashv}} \chi}{f : \phi \mathrel{\dot{\dashv}} \psi \qquad g : \psi \mathrel{\dot{\dashv}} \chi} \qquad \text{(Homo Compose)}$$

The elegance of this and similar rules is the reason to prefer the notation "$f : \phi \mathrel{\dot{\dashv}} \psi$" over "$\phi \cdot f^\dagger = f \cdot \psi$"; unfortunately, not all reasoning can be done in terms of these formulas. (We can even strengthen the rule to an equivalence if we weaken the left conjunct of the bottom line to the assertion "$\phi \cdot f^\dagger = f \cdot \psi$ **on** the range of $g^\dagger$", or, alternatively, if we assume that $g \mathrel{\dot{\dashv}}$ has a right inverse. This is clear from any straightforward attempt to prove the rule.)

There are more properties that are worth to be mentioned (and can or will be used in the sequel). We give some of them.

$$\mathsf{id} : \phi \mathrel{\dot{\dashv}} \phi \qquad \text{(Homo Id)}$$

$$\Leftarrow \frac{f^\dagger : \phi^\dagger \mathrel{\dot{\dashv}} \psi^\dagger}{f : \phi \mathrel{\dot{\dashv}} \psi} \qquad \text{(Homo Functor)}$$

$$f : f \mathrel{\dot{\dashv}} f^\dagger \qquad \text{(Homo Self)}$$

$$\Leftarrow \frac{f : \phi \mathrel{\dot{\dashv}} \psi}{g \cdot f = \mathsf{id} \qquad \phi = f \cdot \psi \cdot g^\dagger} \qquad \text{(Homo Inj)}$$

These are also very easy to check; as an example we give a calculational proof of the surprising rule Homo Inj, asserting that any injective function $f : A \leftarrow B$ is a homomorphism from whatever operation $\psi : B \leftarrow B^\dagger$ you like (to some ϕ).

$$\begin{array}{ll} & f \text{ is a } \dagger\text{-homomorphism, say } f : \phi \mathrel{\dot{\dashv}} \psi \\ \equiv & \quad \text{definition homomorphism} \\ & \phi \cdot f^\dagger = f \cdot \psi \\ \equiv & \quad \text{take } g \text{ to be the left inverse of } f \text{ (i.e., } f \text{ is injective);} \\ & \quad \text{so that } \mathsf{id}_{B^\dagger} = \mathsf{id}_B^\dagger = (g \cdot f)^\dagger = g^\dagger \cdot f^\dagger \end{array}$$

$$
\begin{array}{ll}
 & \phi \cdot f^\dagger = f \cdot \psi \cdot g^\dagger \cdot f^\dagger \\
\Leftarrow & \qquad \text{Leibniz} \\
 & \phi = f \cdot \psi \cdot g^\dagger
\end{array}
$$

The last equation equivales true when we *define* ϕ to be $f \cdot \psi \cdot g^\dagger$. That is, we have proved: $(\exists \phi :: f : \phi \mathrel{\dot{\leftarrow}} \psi) \Leftarrow g \cdot f = \mathsf{id}$.

Because of rules Homo Compose and Homo Id, the $\dagger$-algebras, as objects, with the $\dagger$-homomorphisms, as arrows, form a category. Initial objects in this category will play an important rôle; in the next section we spell out the definition of initiality for this category.

4 Data types and catamorphisms

In categorical terms, a data type is just a $\dagger$-algebra, for any functor $\dagger$. A very important data type is the initial $\dagger$-algebra $\mu(\dagger)$. We shall define it categorically, and then go on to spell out the definition, eliminating categorical terms as much as possible. One of the concepts that we will come across, will be termed catamorphism.

In the theorem below there occurs a condition on functors that we are not going to explain: the second claim says that the functors of interest do satisfy the condition. The theorem (hence the definition following it) is not true in any category; the category has to satisfy some conditions that are met by the category **SET**. Henceforth we assume that we are working in the category **SET**, or any other category that satisfies the conditions.

Theorem 1 *Let $\dagger$ be a ω-co-continuous mono-functor. Then there exists, unique up to isomorphism, a $\dagger$-algebra that is initial in the category of $\dagger$-algebras.*

Theorem 2 *The functors built from the combinators and the "constant functors A" by means of functor composition are all ω-co-continuous, (as is functor $\star$ defined below; see Malcolm[4]).*

Definition 1 *Let $\dagger$ be a ω-co-continuous mono-functor. Then $\mu(\dagger)$ denotes the algebra asserted to exist by the previous theorem.*

We will often name the algebra $\mu(\dagger)$ as $(L, \underline{\mathsf{in}}) = \mu(\dagger)$; it may be helpful to think of L as *L*ist (and it is also mnemonic for *L*eft object, for those familiar with Hagino [1]), and $\underline{\mathsf{in}}$ is to suggest the construction of elements *into* L. When working with just sets as objects, the theorem can be proved by the well-know *term model* construction. This is particularly simple here as there is only one, unary, operation: $\underline{\mathsf{in}}$. (This definition does not cater for non-free initial data types, i.e., objects that are initial in a category where all objects satisfy a given set of equations. A "set of equations" for a $\dagger$-algebra (A, ϕ) is modelled in the current framework by just one equation $E(\phi)$. For example, let $\phi : L \leftarrow \mathbb{1} \Downarrow (A \Downarrow (L \parallel L))$ so that for some nil, tau, cat we have $\phi = \mathsf{nil} \Downarrow (\mathsf{tau} \Downarrow \mathsf{cat})$. We can express that nil is a left identity of cat by substituting $\mathsf{nil}, \mathsf{cat} := \phi \cdot \dot{\uparrow}, \phi \cdot \dot{\uparrow} \cdot \dot{\uparrow}$ in the equation $\mathsf{id} = \mathsf{cat} \cdot \mathsf{nil} \Uparrow \mathsf{id}$. And $f = g \wedge h = j$ is expressed by the single equation $f \parallel h = g \parallel j$. A categorical treatment of this feature is not in the scope of my current abilities.)

Spelling out the definition of $\dagger$-algebra, initiality, and the thus arising term $\dagger$-homomorphism, we get the following definition of $\mu(\dagger)$.

> Let $\dagger$ be a functor. Then $\mu(\dagger)$ is the pair $(L, \underline{\mathsf{in}})$ consisting of a type L and a function $\underline{\mathsf{in}} : L \leftarrow L^\dagger$ such that the following so-called *initiality property* holds:

For any given type A *and function* $\phi : A \leftarrow A^\dagger$ *, the equation*

$$\phi \cdot f^\dagger \;=\; f \cdot \underline{\mathsf{in}} \tag{1}$$

has a unique solution for f *.*

> Recall that we can write the equation more succinctly as $f : \phi \mathrel{\dot{\leftarrow}} \underline{\mathsf{in}}$. It follows from the typing requirements that $f : A \leftarrow L$. Moreover, it can be proved that $(L, \underline{\mathsf{in}})$ exists and is unique up to isomorphism.

For given $\dagger$, A, ϕ we denote the unique solution for f in (1) by $(\!|\phi|\!)_\dagger$. Any such function $(\!|\phi|\!)_\dagger$ is called a $\dagger$-*catamorphism*, or $\dagger$-*cata* for short. For brevity we don't mention the functor when it is clear from the context what functor is meant, thus writing $(\!|\phi|\!)$ and calling it a catamorphism. Notice that equation (1) may be read from right to left as an inductive definition of f: f applied to any value constructed by $\underline{\mathsf{in}}$ from ingredient(s), x say, is defined as operation ϕ applied to the result(s) of $f^\dagger$ on the ingredient(s) x. Thus $f \cdot \underline{\mathsf{in}}$ models the left hand side of a Functional Programming style definition of f that uses *pattern matching*. (The right-to-left order of the above inductive definition is due to our convention that (operations of) the source are placed at the right, and (operations of) the target at the left.)

As an example, take $\dagger$ defined by $X^\dagger = \mathbb{1} \Yup (X \parallel A)$, for a fixed type A. Let $(L, \underline{\mathsf{in}})$ be $\mu(\dagger)$; we will argue briefly that it is (isomorphic to) the algebra of snoc-lists over A. Recall that any function $f : A \leftarrow B \Yup (C \parallel D)$ can be written as $f = g \Ydown h$ with $g : A \leftarrow B$ and $h : A \leftarrow (C \| D)$. So let $\mathsf{nil} \Ydown \mathsf{snoc}$ be $\underline{\mathsf{in}}$; it follows that $\mathsf{nil} : L \leftarrow \mathbb{1}$ and $\mathsf{snoc} : L \leftarrow (L \| A)$. More conventionally written we have the algebra $(L;\ \mathsf{nil} : L \leftarrow \mathbb{1},\ \mathsf{snoc} : L \leftarrow (L \parallel A))$. It is, furthermore, the initiality property that makes this algebra the familiar snoc-list algebra; any element of L can be written $\mathsf{snoc}(\mathsf{snoc}(\ldots \mathsf{snoc}(\mathsf{nil} \parallel x) \ldots \parallel y) \parallel z)$ for $x, \ldots, y, z \in A$. The catamorphisms for this algebra are the left reductions; $(\!|e^\bullet \Ydown \oplus|\!) = \oplus LRED_e$ for $e \in B$, $\oplus : B \leftarrow (B \parallel A)$.

Originally Meertens[5] used the term homomorphism for what we have called catamorphisms, and this is still current practise in BM. True enough each catamorphism $(\!|\phi|\!)_\dagger$ *is* a homomorphism: $(\!|\phi|\!)_\dagger : \phi \mathrel{\dot{\leftarrow}} \underline{\mathsf{in}}$ (where $(L, \underline{\mathsf{in}}) = \mu(\dagger)$), but opposed to $\dagger$-homomorphisms the source algebra is fixed, and so the $\dagger$-catamorphisms form a proper subset of the homomorphisms, which makes them *not* closed under composition: it is in general not true that $(\!|\phi|\!)_\dagger \cdot (\!|\psi|\!)_\dagger$ equals a $(\!|\chi|\!)_\dagger$ for some χ (even not when ϕ and ψ are equal but different from $\underline{\mathsf{in}}$). To overcome the misleading suggestion of some closure property, and to stress the fact that the source algebra is fixed, the term "catamorphism" has been proposed by Meertens[5].

Below we prove that for any $(L, \underline{\mathsf{in}}) = \mu(\dagger)$ the function $\underline{\mathsf{in}} : L \leftarrow L^\dagger$ has a left and right inverse $\mathsf{out} : L^\dagger \leftarrow L$. Furthermore, they are $\dagger$-homomorphisms: $\mathsf{out} : \underline{\mathsf{in}}^\dagger \mathrel{\dot{\leftarrow}} \underline{\mathsf{in}}$ and $\underline{\mathsf{in}} : \underline{\mathsf{in}} \mathrel{\dot{\leftarrow}} \underline{\mathsf{in}}^\dagger$. So $(L, \underline{\mathsf{in}})$ is isomorphic to $(L^\dagger, \underline{\mathsf{in}}^\dagger)$, that is, $(L, \underline{\mathsf{in}}) \simeq (L^\dagger, \underline{\mathsf{in}}^\dagger) = (L, \underline{\mathsf{in}})^\dagger$ and we may say that $(L, \underline{\mathsf{in}})$ is a *fixed point* of $\dagger$. (Moreover it seems to be claimed in the literature that there exists an ordering such that L is the least fixed point of $\dagger$; this then explains the symbol μ: it is often used as a least fixed point operation symbol.) The inverse out decomposes an element from L into the ingredients from which $\underline{\mathsf{in}}$ can (re)construct the element. Using out we can rewrite —if we wish— the "definition with pattern matching" (1) as $f = \phi \cdot f^\dagger \cdot \mathsf{out}$.

Calculation rules for catamorphisms

Quite remarkably we can capture all of the definition of $\mu(\dagger)$ in one rule for the typing, and one for the initiality property. As regards to typing we have, when $(L, \underline{\mathsf{in}}) = \mu(\dagger)$:

$$\boxed{\underline{\mathsf{in}} : L \leftarrow L^\dagger} \qquad \text{(Mu-Type)}$$

The initiality property is fully captured by the following rule. Given that $(L, \underline{\mathsf{in}}) = \mu(\dagger)$,

$$\boxed{\equiv \frac{(\!|\phi|\!) = f}{\phi \cdot f^\dagger = f \cdot \underline{\mathsf{in}} \qquad \text{i.e., } f : \phi \mathrel{\dot{\leftarrow}} \underline{\mathsf{in}}}} \qquad \text{(Catamorphism)}$$

We show that the initiality property follows from this rule. Substituting $f := (\!|\phi|\!)$ and observing that the top line is true, gives the existence of a solution to (1). Assuming that both f and g are solutions to (1), we have by the rule (in the top line) that $f = (\!|\phi|\!) = g$, thus establishing that (1) has at most one solution. On the other hand, the rule follows from the initiality property. Again, the $\Rightarrow$ part expresses the existence of a solution, called $(\!|\phi|\!)$. The $\Leftarrow$ part merely expresses the uniqueness of the solution.

Rule Mu-Type (together with the categorical typing constraints) has a corollary that is worth knowing by heart (and makes a lot of diagrams superfluous):

$$\Leftarrow \frac{(\!|\phi|\!) : A \leftarrow L}{\phi : A \leftarrow A^\dagger} \qquad \text{(Cata Type)}$$

where $(L, \underline{\mathsf{in}}) = \mu(\dagger)$. In this paper we shall verify the well-typedness of the formulas almost always tacitly. As Catamorphism captures all of the definition of $\mu(\dagger)$ (apart from typing), it has a lot of consequences. For example, substituting $f := (\!|\phi|\!)$ gives immediately

$$(\!|\phi|\!) : \phi \mathrel{\dot{\leftarrow}} \underline{\mathsf{in}} \qquad \text{(Cata Self)}$$

and by substituting $f, \phi := \mathsf{id}, \underline{\mathsf{in}}$ we get that

$$(\!|\underline{\mathsf{in}}|\!) = \mathsf{id} \qquad \text{(Cata Id)}$$

As another example, we may read the top line of Catamorphism as saying "f is a catamorphism"; the bottom line gives a (necessary and) sufficient condition: it requires to construct a ϕ satisfying the equation. These and further uses of Catamorphism in calculations will occur in the sequel. There are also two easy corollaries to rule Catamorphism that are worth a rule of their own: the Unique Extension Property

$$\Leftarrow \frac{f = g}{f : \phi \mathrel{\dot{\leftarrow}} \underline{\mathsf{in}} \qquad g : \phi \mathrel{\dot{\leftarrow}} \underline{\mathsf{in}}} \qquad \text{(UEP)}$$

and the so-called Promotion Theorem, here called Promotion′:

$$\Leftarrow \frac{(\!|\phi|\!) = f \cdot (\!|\psi|\!)}{\phi \cdot f^\dagger = f \cdot \psi \qquad \text{i.e., } f : \phi \mathrel{\dot{\leftarrow}} \psi} \qquad \text{(Promotion′)}$$

assuming, of course, that both catas are $\dagger$-catas. The bottom line merely says that f is a homomorphism, in BM speak we say "f is $\phi \mathrel{\dot{\leftarrow\!\!+}} \psi$-promotable". (An operational interpretation of the top line is that f is "promoted" from behind a catamorphism into the computation of the catamorphism itself. We shall later see a form of promotion where f is promoted from being a post-process of a catamorphism into being a pre-process. So, the name "promotion" is a bit unfortunate here; it had better been called "absorption".) Since homomorphisms are closed under composition, as asserted by rule Homo Compose, applications of Promotion$'$ can be chained easily. Notice that the formulas of rule Catamorphism are an instantiation of Promotion$'$ (by choosing $\psi := \underline{\mathsf{in}}$, and noting that $(\!|\underline{\mathsf{in}}|\!) = \mathsf{id}$). Here follows a three step proof of Promotion$'$.

$$
\begin{array}{ll}
 & (\!|\phi|\!) = f \cdot (\!|\psi|\!) \\
\equiv & \quad \text{Catamorphism} \\
 & (f \cdot (\!|\psi|\!)) : \phi \mathrel{\dot{\leftarrow\!\!+}} \underline{\mathsf{in}} \\
\Leftarrow & \quad \text{Homo Compose} \\
 & f : \phi \mathrel{\dot{\leftarrow\!\!+}} \psi \;\wedge\; (\!|\psi|\!) : \psi \mathrel{\dot{\leftarrow\!\!+}} \underline{\mathsf{in}} \\
\equiv & \quad \text{Cata Self} \\
 & f : \phi \mathrel{\dot{\leftarrow\!\!+}} \psi \,.
\end{array}
$$

Actually, rule Promotion$'$ can be strengthened to an equivalence if the bottom line is weakened by rewriting it into "$\phi \cdot f^\dagger = f \cdot \phi$ **on** the range of $(\!|\psi|\!)^\dagger$". This is clear from the remark following rule Homo Compose. (And as we have said, when doing so, we are essentially working in the category **SET**.)

Here is the promised construction of the catamorphism out that is the inverse of $\underline{\mathsf{in}}$. Type considerations alone already suggest the definition:

$$
\begin{array}{ll}
 & \mathsf{out} : L^\dagger \leftarrow L \text{ is a catamorphism, say } (\!|\phi|\!) \\
\equiv & \quad \text{Cata Type} \\
 & \phi : L^\dagger \leftarrow L^{\dagger\dagger} \\
\equiv & \quad \text{functor calculus, assume } \phi = \psi^\dagger \\
 & \psi : L \leftarrow L^\dagger \\
\equiv & \quad \text{Mu-Type, assume } \psi = \underline{\mathsf{in}} \\
 & \mathsf{true} \,.
\end{array}
$$

Hence, in the assumption that $\mathsf{out} = (\!|\underline{\mathsf{in}}^\dagger|\!)$ we have $\mathsf{out} : L^\dagger \leftarrow L$. The claims $\underline{\mathsf{in}} \cdot \mathsf{out} = \mathsf{id}_L$ and $\mathsf{out} \cdot \underline{\mathsf{in}} = \mathsf{id}_{L^\dagger}$ are easily verified by calculation.

$$
\begin{array}{ll}
 & \mathsf{id} = \underline{\mathsf{in}} \cdot \mathsf{out} \\
\equiv & \quad \text{Cata Id, definition } \mathsf{out} \\
 & (\!|\underline{\mathsf{in}}|\!) = \underline{\mathsf{in}} \cdot (\!|\underline{\mathsf{in}}^\dagger|\!) \\
\Leftarrow & \quad \text{Promotion}' \\
 & \underline{\mathsf{in}} : \underline{\mathsf{in}} \mathrel{\dot{\leftarrow\!\!+}} \underline{\mathsf{in}}^\dagger \\
\equiv & \quad \text{Homo Self} \\
 & \mathsf{true}
\end{array}
$$

and conversely, to show that out is a left inverse of $\underline{\mathsf{in}}$:

$$\begin{array}{ll} & \mathsf{out} \cdot \underline{\mathsf{in}} \\ = & \qquad \text{definition } \mathsf{out}\text{, CATAMORPHISM} \\ & \underline{\mathsf{in}}^\dagger \cdot \mathsf{out}^\dagger \\ = & \qquad \text{functor calculus} \\ & (\underline{\mathsf{in}} \cdot \mathsf{out})^\dagger \\ = & \qquad \text{above: } \mathsf{out} \text{ is right inverse of } \underline{\mathsf{in}}\text{, functor calculus} \\ & \mathsf{id}\,. \end{array}$$

The claim $\underline{\mathsf{in}} : \underline{\mathsf{in}} \mathrel{\dot{\Leftarrow}} \underline{\mathsf{in}}^\dagger$ follows immediately by HOMO SELF; $\mathsf{out} = (\!|\underline{\mathsf{in}}^\dagger|\!) : \underline{\mathsf{in}}^\dagger \mathrel{\dot{\Leftarrow}} \underline{\mathsf{in}}$ follows immediately by CATA SELF.

* * *

As a final word about catamorphisms in general, we remind you of rule HOMO INJ: $(\exists \phi :: f : \phi \mathrel{\dot{\Leftarrow}} \psi) \Leftarrow g \cdot f = id$. Taking $(B, \psi) := (L, \underline{\mathsf{in}}) = \mu(\dagger)$ we have by CATAMORPHISM that any injective (i.e., having a left inverse) function $f : A \leftarrow L$ is a $\dagger$-catamorphism. Moreover, for *any* function $f : A \leftarrow L$ we have, by the combinator calculus, that $f \Uparrow \mathsf{id} : A \leftarrow L$ is injective (since id is) and so $f = \ll \cdot (\!|\phi|\!)$ for some ϕ. This is illustrated below for the factorial function.

A worked example: $\mathbb{N}$ and *fac*

The definitions of μ-types and $\dagger$-catamorphisms are quite abstract (if you are not used to it). Here we give an worked example. We start with a definition of the natural numbers $(\mathbb{N}, \underline{\mathsf{in}}) = \mu(\dagger)$ for a suitably chosen functor $\dagger$. Then we give the "standard" recursive equations for the factorial function *fac*, and express them using only combinators and $\underline{\mathsf{in}}$ (and the operations of the target algebra). Finally, remembering what we said just above, we proceed to write $fac \Uparrow \mathsf{id}$ (that is, *fac* tupled with id) as a $\dagger$-catamorphism.

* * *

In a Miranda-like style one would define $\mathbb{N} := \mathsf{zero} \mid \mathsf{succ}\,\mathbb{N}$, or, turning zero into a "nullary" function,

$$\mathbb{N} \;:=\; \mathsf{zero}\,\mathbb{1} \mid \mathsf{succ}\,\mathbb{N}\,.$$

In view of this equation we must have, for the required functor $\dagger$, that $\mathbb{N}^\dagger = \mathbb{1} \Yup \mathbb{N}$. So we define $\dagger$ and then $\mathbb{N}, \underline{\mathsf{in}}$ by

$$\begin{array}{rcl} X^\dagger & = & \mathbb{1} \Yup X\,, \\ (\mathbb{N}, \underline{\mathsf{in}}) & = & \mu(\dagger)\,. \end{array}$$

It follows that $\mathbb{N} \simeq \mathbb{1} \Yup \mathbb{N}$, and $\underline{\mathsf{in}} : \mathbb{N} \leftarrow \mathbb{1} \Yup \mathbb{N}$ is the isomorphism. Recall further that any function $f : A \leftarrow B \Yup C$ can be written as $g \Ydown h$ where $g = f \cdot \Yleft : A \leftarrow B$ and $h = f \cdot \Yright : A \leftarrow C$. So define

$$\begin{array}{rcll} \mathsf{zero} & = & \underline{\mathsf{in}} \cdot \Yleft & : \mathbb{N} \leftarrow \mathbb{1} \\ \mathsf{succ} & = & \underline{\mathsf{in}} \cdot \Yright & : \mathbb{N} \leftarrow \mathbb{N} \end{array}$$

and we have $\underline{\mathsf{in}} = \mathsf{zero} \Ydown \mathsf{succ}$.

Now consider the factorial function $fac : \mathbb{N} \leftarrow \mathbb{N}$. For the specification of fac we use $(\mathbb{N}, \underline{\mathsf{in}})$ as the source algebra (where each element is uniquely constructable by $\underline{\mathsf{in}} = \mathsf{zero} \Downarrow \mathsf{succ}$), and we use $(\mathbb{N}; 0, 1, +, \times, \ldots)$ as the target algebra (which facilitates a much simpler specification than using $(\mathbb{N}, \underline{\mathsf{in}})$ again; in particular so when we would assume that ! belongs to the operations...). A Miranda-like equation for fac reads:

$$\begin{array}{lcl} fac \cdot \mathsf{zero} \cdot \iota & = & 1\,, \\ fac \cdot \mathsf{succ} \cdot p & = & (fac \cdot p) \times (p{+}1)\,. \end{array}$$

(We have identified 'constants' with 'functions from $\mathbb{1}$'.) Rewriting the equations using $\underline{\mathsf{in}}$ gives

$$\begin{array}{lcll} fac \cdot \underline{\mathsf{in}} \cdot \Downarrow \cdot \iota & = & 1 & = 1^\bullet \cdot \iota\,, \\ fac \cdot \underline{\mathsf{in}} \cdot \Downarrow \cdot p & = & (fac\, p) \times (p{+}1) & = \times \cdot fac \Uparrow +1 \cdot p \end{array}$$

or, as one functional identity,

$$fac \cdot \underline{\mathsf{in}} \;=\; 1^\bullet \Downarrow (\times \cdot fac \Uparrow +1)\,.$$

In order to write fac itself as a $\dagger$-catamorphism $(\!|\psi|\!)$ on account of Catamorphism, we need an equation of the form $fac \cdot \underline{\mathsf{in}} = \psi \cdot fac^\dagger$. Such a ψ doesn't seem easy to obtain. But now, recall that any tupling with id does give a catamorphism. Indeed, it *is* easy to obtain an equation of the form $(fac \Uparrow \mathsf{id}) \cdot \underline{\mathsf{in}} = \psi \cdot (fac \Uparrow \mathsf{id})^\dagger$, for we have on the one hand (using only combinator and functor calculus)

$$\begin{array}{lcl} fac \cdot \underline{\mathsf{in}} & = & 1^\bullet \Downarrow (\times \cdot fac \Uparrow +1) \\ & = & 1^\bullet \Downarrow (\times \cdot \mathsf{id} \Uparrow +1) \cdot \mathsf{id} \Downarrow (fac \Uparrow \mathsf{id}) \\ & = & 1^\bullet \Downarrow (\times \cdot \mathsf{id} \Uparrow +1) \cdot (fac \Uparrow \mathsf{id})^\dagger \\ & = & \phi \cdot (fac \Uparrow \mathsf{id})^\dagger\,, \end{array}$$

where we define ϕ as suggested, and on the other hand,

$$\begin{array}{lcl} \mathsf{id} \cdot \underline{\mathsf{in}} & = & \underline{\mathsf{in}} \cdot \mathsf{id}^\dagger \qquad \text{(from Catamorphism and Cata Id)} \\ & = & \underline{\mathsf{in}} \cdot (\gg \cdot fac \Uparrow \mathsf{id})^\dagger \\ & = & \underline{\mathsf{in}} \cdot \gg^\dagger \cdot (fac \Uparrow \mathsf{id})^\dagger\,. \end{array}$$

Tupling fac with id gives, again using only combinator and functor calculus,

$$\begin{array}{ll} & fac \Uparrow \mathsf{id} \cdot \underline{\mathsf{in}} \\ = & \\ & (fac \cdot \underline{\mathsf{in}}) \Uparrow (\mathsf{id} \cdot \underline{\mathsf{in}}) \\ = & \qquad \text{above equations for } fac \text{ and } \mathsf{id} \\ & (\phi \cdot (fac \Uparrow \mathsf{id})^\dagger) \;\Uparrow\; (\underline{\mathsf{in}} \cdot \gg^\dagger \cdot (fac \Uparrow \mathsf{id})^\dagger) \\ = & \\ & \phi \Uparrow (\underline{\mathsf{in}} \cdot \gg^\dagger) \cdot (fac \Uparrow \mathsf{id})^\dagger \end{array}$$

and so we have what we were aiming at:

$$\begin{array}{ll} & \mathsf{true} \\ \equiv & \qquad \text{above calculated} \\ & \phi \Uparrow (\underline{\mathsf{in}} \cdot \gg^\dagger) \cdot (fac \Uparrow \mathsf{id})^\dagger \;=\; (fac \Uparrow \mathsf{id}) \cdot \underline{\mathsf{in}} \\ \equiv & \qquad \text{Catamorphism} \end{array}$$

$$
\begin{array}{ll}
 & fac \Uparrow \mathsf{id} = (\!|\phi \Uparrow (\underline{\mathsf{in}} \cdot \gg^{\dagger})|\!) \\
\Rightarrow & \qquad \text{Leibniz} \\
 & \ll \cdot\, fac \Uparrow \mathsf{id} = \ll \cdot\, (\!|\phi \Uparrow (\underline{\mathsf{in}} \cdot \gg^{\dagger})|\!) \\
\equiv & \qquad \text{combinator calculus} \\
 & fac = \ll \cdot\, (\!|\phi \Uparrow (\underline{\mathsf{in}} \cdot \gg^{\dagger})|\!)\,.
\end{array}
$$

This is exactly the result that is predicted by Meertens[7] in his paper on paramorphisms. (Indeed, nowhere we have used any specific properties of ϕ and the construction is completely general.)

5 Reduction and map

There are two forms of catamorphisms that turn up over and over again and have very handy calculational rules, namely —sometimes unconditional— functional identities. These are known as *reduction* and *map*. As we define the notion of map and reduction in a very general way, we shall take care that they agree with what we are used to for the familiar data types, and, in particular, satisfy the laws that we associate to them. The consequence is that we can not define reductions for any data type, though we can do so for maps. Part of this section is heavily inspired by Verwer[8] and Malcolm[3].

It is basic to the notion of reduction and map that one can speak of a parameter type of the μ-type. To be able to do so we use bi-functors, written as normal infix operations. We shall consider its right argument as "the parameter type" and use this terminology. Notice that for any mono-functor $\ddagger$ we can define a bi-functor $\dagger$ that doesn't use its right argument and so behaves as $\ddagger$:

$$
\begin{array}{lcl}
X \dagger A & = & X^{\ddagger}\,, \\
f \dagger g & = & f^{\ddagger}
\end{array}
$$

(which defines a bi-functor indeed). Therefore, the use of bi-functors is without loss of generality: given some $\ddagger$ we can everywhere replace $X^{\ddagger}$ by $X \dagger \mathbb{1}$, and $f^{\ddagger}$ by $f \dagger \mathsf{id}$.

Now, let $\dagger$ be a bi-functor. By fixing the parameter type, to A say, we get a mono-functor $\dagger A$:

$$
\begin{array}{lcl}
X^{\dagger A} & = & X \dagger A\,, \\
f^{\dagger A} & = & f \dagger \mathsf{id}_A\,.
\end{array}
$$

(These definitions are consistent with considering $\dagger A$ as mapping any x into $x^{\mathsf{id}} \dagger x^{A}$ and considering A itself as a constant functor with $X^{A} = A$, $f^{A} = \mathsf{id}_A$.) (As an example you might think of $X \dagger A = \mathbb{1} \Yup (X \parallel A)$, so that $\dagger A$ is the functor for snoc lists over A discussed earlier, but now the parameter type has been made explicit as its second argument. If we further define $f \dagger g = \mathsf{id} \Yup (f \parallel g)$, then we see that g acts on the elements of the parameter type. So, for these kind of functors the notion of map, defined below, agrees with what we are used to.)

We shall be considering μ-types for mono-functors $\dagger A$ and $\dagger B$ and so on. As an abbreviation, let $A^{\star}$ denote the corresponding L_A, i.e.,

$$
A^{\star} \quad = \quad L_A \quad \text{where } (L_A, \underline{\mathsf{in}}_A) = \mu(\dagger A)\,.
$$

Thus $\star$ depends on $\dagger$ but we shall nowhere make this explicit in the notation. (Actually, we would like to use $\dagger$ instead of $\star$, thereby overloading the bi-functor symbol (and suggesting

the dependency *very* clearly), but we shall not do so here.) This definition of $\star$ will be valid throughout this section.

Map

Our goal is to define, for suitable function f, a function $f^\star$ (f-map). There are a few requirements to call $f^\star$ a map function: First, we want $f^\star : A^\star \leftarrow B^\star$ whenever $f : A \leftarrow B$. Second, we want $f^\star$ to be a $\dagger B$-catamorphism. Third, we require the following laws to be true:

$$\begin{array}{rcll} \mathsf{id}_A^\star & = & \mathsf{id}_{A^\star} & \text{(Map Id)} \\ (f \cdot g)^\star & = & f^\star \cdot g^\star & \text{(Map Distribution)} \end{array}$$

(In other words, these laws together with the typing requirement, assert that $\star$ is a functor!) Finally, for familiar data types the newly defined maps should agree with the familiar maps.

The type requirements do not leave much room to define $f^\star$:

$$\begin{array}{ll} & f^\star : A^\star \leftarrow B^\star \text{ is a } \dagger B\text{-catamorphism, say } (\!|\phi|\!) \\ \equiv & \quad \text{Cata Type} \\ & \phi : A^\star \leftarrow A^\star \dagger B \\ \equiv & \quad \text{recall } f : A \leftarrow B \text{, assume } \phi = \psi \cdot \mathsf{id} \dagger f \\ & \psi : A^\star \leftarrow A^\star \dagger A \\ \equiv & \quad \text{recall } \underline{\mathsf{in}}_A : A^\star \leftarrow A^\star \dagger A \text{, assume } \psi = \underline{\mathsf{in}} \\ & \mathsf{true}\,. \end{array}$$

Hence assuming

$$f^\star \quad = \quad (\!|\underline{\mathsf{in}} \cdot \mathsf{id} \dagger f|\!)_{\dagger B} \qquad \text{(Map)}$$

we have proven the type requirement. (The definition of $f^\star$ depends on A and B; this is allowed for a functor $\star$, since formally any functor $\dagger$ consists of a mapping $\dagger_{\mathrm{tp}}$ from objects to objects together with a family of mappings $\dagger_{A,B}$ from arrows $f : A \leftarrow B$ to arrows $f^{\dagger_{A,B}} : A^{\dagger_{\mathrm{tp}}} \leftarrow B^{\dagger_{\mathrm{tp}}}$. We have consistently omitted these subscripts to the functors, and even not mentioned the dependency in the definition of a category.) It is easy to verify the two laws; e.g., for Map Distribution we calculate

$$\begin{array}{ll} & (f \cdot g)^\star = f^\star \cdot g^\star \\ \equiv & \quad \text{Map} \\ & (\!|\underline{\mathsf{in}} \cdot \mathsf{id} \dagger (f \cdot g)|\!) = f^\star \cdot (\!|\underline{\mathsf{in}} \cdot \mathsf{id} \dagger g|\!) \\ \Leftarrow & \quad \text{Promotion}' \\ & \underline{\mathsf{in}} \cdot \mathsf{id} \dagger (f \cdot g) \cdot f^\star \dagger \mathsf{id} = f^\star \cdot \underline{\mathsf{in}} \cdot \mathsf{id} \dagger g \\ \equiv & \quad \text{Map \& Catamorphism in rhs} \\ & \underline{\mathsf{in}} \cdot \mathsf{id} \dagger (f \cdot g) \cdot f^\star \dagger \mathsf{id} = \underline{\mathsf{in}} \cdot \mathsf{id} \dagger f \cdot f^\star \dagger \mathsf{id} \cdot \mathsf{id} \dagger g \\ \equiv & \quad \text{functor calculus} \\ & \mathsf{true}\,. \end{array}$$

There are some more useful laws that we associate with maps. We give two of them.

$$\Leftarrow \frac{(\!|\phi|\!) \cdot f^\star = (\!|\psi|\!)}{\phi \cdot \mathsf{id} \dagger f = \psi} \quad \text{i.e.,} \quad (\!|\phi|\!) \cdot f^\star = (\!|\phi \cdot \mathsf{id} \dagger f|\!) \qquad (\textsc{Cata Factorisation}')$$

$$\Leftarrow \frac{(\!|\phi|\!) \cdot f^\star = f \cdot (\!|\psi|\!)}{\phi \cdot f \dagger f = f \cdot \psi} \quad \text{i.e.,} \quad \Leftarrow \frac{f : (\!|\phi|\!) \stackrel{\star}{\leftarrow} (\!|\psi|\!)}{f : \phi \stackrel{\ddagger}{\leftarrow} \psi} \qquad (\textsc{Promotion})$$

The first of these is Verwer's[8] Factorisation Theorem: in his terminology a reduction $\oplus/$ is just the same as $(\!|\oplus|\!)$ except for the additional typing requirement that $\oplus : A \leftarrow A \dagger A$ rather than $\oplus : A \leftarrow A \dagger B$. Our proof does not use this typing anywhere, nor does Verwer's proof. Unless these catamorphisms $(\!|\oplus|\!)$ turn out to have some nice properties not shared by catamorphisms in general, we feel that they are not worth a specific name.

The top line of the second rule expresses that f is "promoted" from being a post-process of a catamorphism into being a pre-process. Such a promotion can have a drastical effect on the computation time. As regards to the bottom line of the alternative formulation of Promotion, we define for a bi-functor $\dagger$ that $f : \phi \stackrel{\ddagger}{\leftarrow} \psi$ means $\phi \cdot f \dagger f = f \cdot \psi$. In this context, the equation is usually read as "f is $\phi \stackrel{\ddagger}{\leftarrow} \psi$-distributive". (Thus in the absence of laws, promotability and distributivity are really the same concepts; 'promotability' is used with respect to a mono-functor $\star$ while 'distributivity' is used with respect to a bi-functor $\dagger$.)

The proofs are simple calculations. For Cata Factorisation$'$ we have:

$$\begin{array}{ll}
& (\!|\phi|\!) \cdot f^\star = (\!|\psi|\!)_{\dagger A} \\
\equiv & \quad \text{Catamorphism (interchanging left and right hand sides)} \\
& (\!|\phi|\!) \cdot f^\star \cdot \underline{\mathsf{in}} = \psi \cdot ((\!|\phi|\!) \cdot f^\star)^{\dagger A} \\
\equiv & \quad \text{at the left: apply first Map\&Catamorphism then Catamorphism once more} \\
& \phi \cdot (\!|\phi|\!) \dagger \mathsf{id} \cdot f^\star \dagger f = \psi \cdot ((\!|\phi|\!) \cdot f^\star)^{\dagger A} \\
\equiv & \quad \text{at the right: functor calculus} \\
& \phi \cdot (\!|\phi|\!) \dagger \mathsf{id} \cdot f^\star \dagger f = \psi \cdot (\!|\phi|\!) \dagger \mathsf{id} \cdot f^\star \dagger \mathsf{id} \\
\Leftarrow & \quad \text{functor calculus; Leibniz} \\
& \phi \cdot \mathsf{id} \dagger f \,.\, = \psi \,.
\end{array}$$

For Promotion we calculate:

$$\begin{array}{ll}
& (\!|\phi|\!) \cdot f^\star = f \cdot (\!|\psi|\!) \\
\equiv & \quad \textsc{Cata Factorisation}' \\
& (\!|\phi \cdot \mathsf{id} \dagger f|\!) = f \cdot (\!|\psi|\!) \\
\Leftarrow & \quad \textsc{Promotion}' \\
& \phi \cdot \mathsf{id} \dagger f \cdot f \dagger \mathsf{id} = f \cdot \psi \\
\equiv & \quad \text{functor calculus} \\
& \phi \cdot f \dagger f = f \cdot \psi \,.
\end{array}$$

Reduction

Let us now define reductions. Apart from the explicit control over the parameter type, we must also have some control over the functor in order to express the desired properties.

The requirements for $\oplus/$ to be called a reduction read as follows. First, it should agree with the well-known reduction, when specialised to the data type of lists. Second, we want $\oplus/ : A \leftarrow A^\star$ (reduction "reduces" one $\star$). Third, we wish $\oplus/$ be a catamorphism, c.q. a $\dagger A$-catamorphism, and, fourth, we want *at least* the following laws be valid:

$$\begin{aligned} \oplus/ \cdot \mathsf{tau} &= \mathsf{id} && \text{(Red One Point)} \\ \oplus/ \cdot (\oplus/ \cdot f^\star)^\star &= (\oplus/ \cdot f^\star) \cdot \mathsf{join}/ && \text{(Cata Promotion}') \end{aligned}$$

Thus we must be able to define something like tau and join. This is possible for the data types that are *factorable* as defined by Malcolm[2], i.e., defined as the μ-type of a bi-functor $\dagger$ for which there exists a mono-functor $\ddagger$ such that

$$X \dagger A \;=\; X^\ddagger \Downarrow A \,.$$

(So the functor for snoc-lists is not factorable.) For factorable functors we have explicit control over what happens to the ingredients that come from the parameter type. Indeed, now we can define

$$\begin{aligned} \mathsf{tau} = \;& \underline{\mathsf{in}} \cdot \Uparrow \;\; : A^\star \leftarrow A && \text{(Tau)} \\ \mathsf{join} = \;& \underline{\mathsf{in}} \cdot \Downarrow \;\; : A^\star \leftarrow A^{\star\ddagger} && \text{(Join)} \end{aligned}$$

so that the laws above make sense. Any $\phi : B \leftarrow B \dagger A$ can now be written as $\phi = \oplus \Downarrow f$ for some $\oplus : B \leftarrow B^\ddagger$ and $f : B \leftarrow A$ (actually, $f = (\!|\phi|\!) \cdot \mathsf{tau}$). We will call $\oplus$ the *operation of* the catamorphism $(\!|\oplus \Downarrow f|\!)$. (Notice also that, for a factorable functor $\dagger$ a map $f^\star$ equals $(\!|\mathsf{join} \Downarrow f|\!)$.)

As was the case with maps, the type requirements alone already suggest the definition of reductions to a large extent:

$$\begin{aligned} & \oplus/ : A \leftarrow A^\star \;\text{ is a } \dagger A\text{-catamorphism } (\!|\phi|\!) \\ \equiv \;& \quad \text{Cata Type} \\ & \phi : A \leftarrow A \dagger A \\ \equiv \;& \quad \text{factorability } \dagger \\ & \phi : A \leftarrow A^\ddagger \Downarrow A \\ \equiv \;& \quad \text{combinator calculus, assume } \phi = \psi \Downarrow \mathsf{id} \text{ (motivated below)} \\ & \psi : A \leftarrow A^\ddagger \\ \equiv \;& \quad \text{assume } \psi = \oplus : A \leftarrow A^\ddagger \\ & \mathsf{true} \,. \end{aligned}$$

We have taken id as the second branch of ϕ in order that it acts as the identity on the parameter type, cf. rule Red One Point. Thus we define for $\oplus : A \leftarrow A^\ddagger$

$$\oplus/ \;=\; (\!|\oplus \Downarrow \mathsf{id}|\!)_{\dagger A} \qquad \text{(Reduction)}$$

and it turns out that we can prove all the required properties and some more:

$$\begin{aligned} \oplus/ \cdot f^\star = (\!|\phi|\!) \qquad & \text{where } \oplus \Downarrow f = \phi \qquad \text{(Cata Factorisation)} \\ \oplus/ \cdot (\!|\phi|\!)^\star \;=\; (\!|\phi|\!) \cdot \mathsf{join}/ \qquad & \text{where } \oplus \Downarrow f = \phi \qquad \text{(Cata Promotion)} \end{aligned}$$

Rule Cata Factorisation is just Malcolm's[3] Factorisation Theorem. On the one hand it says that any catamorphism $(\!|\phi|\!)$ can be factored in a reduction after a map by choosing $\oplus := \phi \cdot \natural$ and $f := \phi \cdot \natural^{\natural} = (\!|\phi|\!) \cdot \mathsf{tau}$. On the other hand it says how to construct ϕ when $\oplus$ and f are given: $\phi := \oplus \Downarrow f$. Before delving into the proofs we first observe that, in view of Cata Factorisation, rule Cata Promotion is equivalent to Cata Promotion′. The Cata Promotion rules have two useful corollaries: by specialising $(\!|\phi|\!)$ to $f^\star$ (i.e., $\oplus := \mathsf{join}$) respectively to $\oplus/$ (i.e., $f := \mathsf{id}$) we obtain

$$\begin{array}{rcll} \mathsf{join}/ \cdot f^{\star\star} & = & f^\star \cdot \mathsf{join}/ & \text{(Map Promotion)} \\ \oplus/ \cdot \oplus/^\star & = & \oplus/ \cdot \mathsf{join}/ & \text{(Red Promotion)} \end{array}$$

Recall that we can write Map Promotion more succinctly as $f^\star : \mathsf{join}/ \stackrel{\star}{\leftarrow} \mathsf{join}/$; but we can also say, using the notation of category theory for *natural transformation*, $\mathsf{join}/ : \star\star \stackrel{\bullet}{\leftarrow} \star$.

Here are the proofs. First Red One Point.

$$\begin{array}{ll} & \oplus/ \cdot \mathsf{tau} \\ = & \quad \text{Reduction, Tau} \\ & (\!|\oplus \Downarrow \mathsf{id}|\!) \cdot \underline{\mathsf{in}} \cdot \natural \\ = & \quad \text{Catamorphism} \\ & \oplus \Downarrow \mathsf{id} \cdot (\oplus \Downarrow \mathsf{id})^{\dagger A} \cdot \natural \\ = & \quad \text{factorability } \dagger \\ & \oplus \Downarrow \mathsf{id} \cdot (\oplus \Downarrow \mathsf{id})^{\ddagger} \Uparrow \mathsf{id} \cdot \natural \\ = & \quad \text{functor and combinator calculus} \\ & \mathsf{id}\,. \end{array}$$

Next we prove Malcolm's Cata Factorisation.

$$\begin{array}{ll} & (\!|\phi|\!) = \oplus/ \cdot f^\star \\ \equiv & \quad \text{Reduction, Cata Factorisation}' \\ & (\!|\phi|\!) = (\!|\oplus \Downarrow \mathsf{id} \cdot \mathsf{id} \dagger f|\!) \\ \Leftarrow & \quad \text{Leibniz} \\ & \phi = \oplus \Downarrow \mathsf{id} \cdot \mathsf{id} \dagger f \\ \equiv & \quad \text{factorability } \dagger \\ & \phi = \oplus \Downarrow \mathsf{id} \cdot \mathsf{id}^{\ddagger} \Uparrow f \\ \equiv & \quad \text{combinator and functor calculus} \\ & \phi = \oplus \Downarrow f \end{array}$$

Finally, we prove Cata Promotion. Let $\phi = \oplus \Downarrow f$.

$$\begin{array}{ll} & \oplus/ \cdot (\!|\phi|\!)^\star \;=\; (\!|\phi|\!) \cdot \mathsf{join}/ \\ \equiv & \quad \text{at the left and right: Reduction} \\ & (\!|\oplus \Downarrow \mathsf{id}|\!) \cdot (\!|\phi|\!)^\star \;=\; (\!|\phi|\!) \cdot (\!|\mathsf{join} \Downarrow \mathsf{id}|\!) \\ \equiv & \quad \text{at the left: Cata Factorisation}' \\ & (\!|\oplus \Downarrow \mathsf{id} \cdot \mathsf{id}^{\ddagger} \Uparrow (\!|\phi|\!)|\!) \;=\; (\!|\phi|\!) \cdot (\!|\mathsf{join} \Downarrow \mathsf{id}|\!) \\ \Leftarrow & \quad \text{Promotion}' \text{ and factorability } \dagger \end{array}$$

$$
\begin{array}{ll}
 & \oplus \Downarrow \mathsf{id} \cdot \mathsf{id}^{\ddagger} \Uparrow (\!|\phi|\!) \cdot (\!|\phi|\!)^{\ddagger} \Uparrow \mathsf{id} \;=\; (\!|\phi|\!) \cdot \mathsf{join} \Downarrow \mathsf{id} \\
\equiv & \quad \text{at the left: combinator calculus} \\
 & (\oplus \cdot (\!|\phi|\!)^{\ddagger}) \Downarrow (\!|\phi|\!) \;=\; (\!|\phi|\!) \cdot \mathsf{join} \Downarrow \mathsf{id} \\
\equiv & \quad \text{at the right: JOIN and combinator calculus} \\
 & (\oplus \cdot (\!|\phi|\!)^{\ddagger}) \Downarrow (\!|\phi|\!) \;=\; ((\!|\phi|\!) \cdot \underline{\mathsf{in}} \cdot \curlywedge) \Downarrow (\!|\phi|\!) \\
\equiv & \quad \text{at the right: CATAMORPHISM and factorability } \dagger \\
 & (\oplus \cdot (\!|\phi|\!)^{\ddagger}) \Downarrow (\!|\phi|\!) \;=\; (\phi \cdot (\!|\phi|\!)^{\ddagger} \Uparrow \mathsf{id} \cdot \curlywedge) \Downarrow (\!|\phi|\!) \\
\equiv & \quad \text{at the right: combinator calculus} \\
 & (\oplus \cdot (\!|\phi|\!)^{\ddagger}) \Downarrow (\!|\phi|\!) \;=\; (\phi \cdot \curlywedge \cdot (\!|\phi|\!)^{\ddagger}) \Downarrow (\!|\phi|\!) \\
\Leftarrow & \quad \text{Leibniz} \\
 & \oplus = \phi \cdot \curlywedge \\
\equiv & \quad \text{given: } \phi = \oplus \Downarrow f \\
 & \mathsf{true}\,.
\end{array}
$$

* * *

Now that we have defined tau, we can also claim

$$f^{\star} \cdot \mathsf{tau} \;=\; \mathsf{tau} \cdot f\,. \qquad \text{(MAP ONE POINT)}$$

and also (cf. RED ONE POINT)

$$\mathsf{join}/ \cdot \mathsf{tau}^{\star} \;=\; \mathsf{id} \qquad (\bigcirc)$$

(The rules $(\bigcirc)$, CATA PROMOTION (with $f := \mathsf{id}$ and $\oplus := \mathsf{join}$), and RED ONE POINT together assert that $(\star, \mathsf{tau}, \mathsf{join})$ is a so-called *monad*.)

References

[1] T. Hagino. A typed lambda calculus with categorical type constructors. In D.H. Pitt, A. Poigné, and D.E. Rydeheard, editors, *Category Theory and Computer Science*, volume 283 of *Lect. Notes in Comp. Sc.*, pages 140–157. Springer Verlag, 1987.

[2] G. Malcolm. Factoring homomorphisms. In *International Summer School on Constructive Algorithmics*, 1989. Held on Ameland, The Netherlands, September 1989. Also Technical Report Computer Science Notes CS 8908, Dept of Math and Comp Sc, University of Groningen, 1989.

[3] G. Malcolm. Homomorphisms and promotability. In J.L.A. van de Snepscheut, editor, *Mathematics of Program Construction*, volume 375 of *Lect. Notes in Comp. Sc.*, pages 335–347. Springer Verlag, 1989.

[4] G. Malcolm. *Algebraic Data Types and Program Transformation*. PhD thesis, University of Groningen, The Netherlands, 1990.

[5] L. Meertens. Algorithmics — towards programming as a mathematical activity. In J.W. de Bakker and J.C. van Vliet, editors, *Proceedings of the CWI Symposium on Mathematics and Computer Science*, pages 289–334. North-Holland, 1986.

[6] L. Meertens. Constructing a calculus of programs. In J.L.A. van de Snepscheut, editor, *Mathematics of Program Construction*, volume 375 of *Lect. Notes in Comp. Sc.*, pages 66–90. Springer Verlag, 1989.

[7] L. Meertens. Paramorphisms. *Formal Aspects of Computing*, 4(5):413–424, 1990.

[8] N. Verwer. Homomorphisms, factorisation and promotion. *The Squiggolist*, 1(3):45–54, 1990. Also technical report RUU-CS-90-5, Utrecht Univerity, 1990.